## REMARKS

This paper responds to the Final Office Action mailed August 2, 2010 for the above application in which claims 144, 145, 155-158, 160, 164-165, 167-172 and 1615 are pending. Reconsideration of the application and claims in light of the foregoing amendments and following is requested. Claims 144, 158, 160, 164, 165, 167 and 1615 have been amended.

***Response to Objection to Claim 144***

With respect to the phrase "being each being defined" it is respectfully noted that correction was made in the prior response filed July 19, 2010. Withdrawal of that objection is respectfully requested.

With respect to the "defined by" language, while the objection is emphatically disagreed with because things most certainly can be defined by what they do. For example, one could have two different types of machines. Both are machines, but one is made for cutting grass and the other for attaching piles of paper together, the first is a lawn mower "type" machine and the second is a "stapler" type of machine. The two are both machines, but one is one type and the other is a different type and both are "defined by" what they do. Nevertheless, claim 144 has been amended to remove that "defined by" language and, consistent with the specification, specify that what was previously referred to as "second type" processing resources are computational processing resources.

In light of the above, withdrawal of the objections to claim 144 is respectfully requested.

***Response to Claim Rejections Under 35 U.S.C. §112, ¶1***

Claims 144-145, 155-158, 160, 164-165 and 167-172 were rejected under 35 U.S.C. §112, ¶1 because certain claim language allegedly lacked written description.

First, the Office Action asserts that the claim 144 language stating "the processing resources of the second type each being defined by being configured to only receive and execute instructions that were delegated to them by processing resources of the first type" lacks written description support.

Second, the Office Action asserts that limitation of claim 1615 stating "receiving an execution instruction of a different thread at the second IPU; determining that the execution instruction of the different thread can not be processed by the second IPU but can be processed

by the other type elemental processing resource is shared by the second IPU and the first IPU" lacks written description support.

In both cases, Applicant respectfully disagrees and will further explain the support for the current claims with reference to various paragraphs of the published application. In addition, it is respectfully submitted that the following walkthrough of most of the paragraphs relevant to obtain an understanding of the subject matter of the instant claims will establish that the currently pending claims are fully supported and also clearly distinguish over the cited references.

Note specifically, that the entire specification only describes the shared computational resources as receiving delegated instructions from first type processing resources for execution. Nowhere is there a statement that the shared computational resources receive instructions via any other way or from any other source. Patent specifications are supposed to describe the way something works. As here, the Applicant has chosen to claim this aspect as it is described, which necessarily means that the specification supports the claim. That is what is being done for the objected-to aspect of the claims.

For the convenience of the Examiner, quotes from the application are identified by normal and/or bolded text (for emphasis) and explanation is provided in italics. To the extent explanation of a particular paragraph or figure is not provided, this is because the text itself is believed to be clear and self explanatory itself or in light of other explanation provided.

> **[0022]** FIG. 9 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing **a resource delegation process in an integer processing unit in which the integer processing unit constructs and delegates a request to other processing resources**;
>
> **[0023]** FIG. 10 is of a block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a form of **a delegation request constructed at a requesting integer processing unit and directed to a mathematical processing unit**;
>
> **[0025]** FIG. 12 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of **request handling in a router of a processor routing requests between the requesting processing resources and servicing processing resources**;

**[0027]** FIG. 14 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure of **routing a result from a target processing resource to delegating processing resources**

**[0053]** FIG. 2 is of a functional block diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing an architecture of Processor 100 of the present invention. **Processor 100 comprises Integer Processing Units ("IPUs") 102, 104, 106, 108, a Router 110, a secondary (i.e., level two) cache memory ("L2 cache") 112 and a Mathematical Processing Unit ("MPU") 114.** The IPUs are disposed in communication with the L2 cache and the MPU via the Router along with System Buses 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148. **In this exemplary architecture, four IPUs 102, 104, 106, 108 are collaborating with single MPU 112 and L2 cache 114 (i.e., a single MPU 112 and L2 cache 114 are shared by four IPUs 102, 104, 106, 108). Alternatively, the number of IPUs, MPUs and L2 caches may vary in other embodiments depending on the requirement of specific applications.** *In other words, this paragraph shows an example of 4 processing resources of the First Type, called IPUs, all connected via an interconnection network, in this case a router, to a shared processing resources of a different type, a computational type called an MPU, which performs delegated computation along with a shared L2 cache and memory/device interface which also incorporates a computational type processor that does delegated atomic memory computation.*

*Paragraph [0053] describes the processor of Fig. 2 as comprised of 4 IPUs collaborating with 1 MPU and 1 L2 cache via a router, making clear that the numbers of IPUs can vary. The MPU is not described as collaborating with the L2 cache –as will be evident the IPUs are in control of both.*

**[0054]** The configuration of Processor 100 will depend on the context of system deployment. Factors such as, but not limited to, the desired throughput of a processor, capacity and/or location of the underlying system, nature of desired system deployment (e.g., portable, desktop, server, and/or the like environments) may affect deployment requirements and configuration. **It should be further noted that disparate processing resources may be employed in various quantities. Processing resources may include digital signal processing units (DSPs), graphic processing units (GPUs), IPUs, input/output controller processing units, memory management units (MMUs), MPUs, processing cache memory, vector processing units (VPUs), and/or the like. For example, in certain embodiments such as 3D animation rendering, floating-point throughput is desirable and Processor 100 may be configured with a single IPU and four MPUs interconnected by a Router. In yet another example embodiment, such as for a high traffic web server, handling multiple web requests simultaneously may require Processor 100 configured with 16 IPUs, a single MPU, and larger processing cache interconnected by a Router.** Further, in certain environments, processing resources may employ fewer or greater than 32-bit addressing and instruction pathways. For example, for

environments where working with large datasets, 64-bit instruction and addressing may be employed for any of the processing resources. *This paragraph thus describes that the IPUs and MPUs of the previous paragraph are only two examples of processing resources, and gives other examples that amount to subparts of a processor, and describes that a processor can be comprised of various quantities of several different kinds of processing resources connected via a Router.*

**[0055] It should be noted that although processing resources on Processor 100 may be external to each other, the architecture and arrangement of the processing resources makes them intra-chip dependent and interactive.** This architecture is significantly different from current and prior art designs. Rather than merely duplicating the functionality of an existing, discrete, monolithic microprocessor design for a system to achieve a multiprocessor system a radical rethinking and implementation is exhibited by Processor 100 of the present invention. **In other words, elemental processing resources (e.g., IPUs, MPUs, L2 caches), which individually may not be capable of servicing an entire instruction set, are dependant on one another. These intra-dependent processing resources, together, service an instruction set more efficiently. Further, through instruction set design and intra-chip networking via a Router, these intra-dependent processing resources are capable of identifying other processing resources to which they may delegate instruction signals that they themselves may be incapable of servicing.** This intra-dependent, networked, processing resource, design chip architecture is more efficient than current and prior art monolithic designs. For example, prior art systems may combine two Intel Pentium chips in a system for extra integer performance. However, such a system will be inefficient as such monolithic chip duplication will also duplicate extra mathematical processing logic, memory management logic, etc., which may go underutilized. Furthermore, such prior art designs depend on processor intensive and time expensive operating system scheduling to make use of the added processing facilities. **Processor 100 of the present invention does not suffer from these limitations. It may employ numerous IPUs to enhance integer processing throughput, and itself may delegate instruction processing to the proper processing resources without duplicating underutilized portions of a chip.** *This paragraph describes that the innovation in chip architecture achieves a more efficient multiprocessor system and can service an instruction set more efficiently by decomposing a conventional monolithic processor design into its intra-dependent elemental processing resources to which instructions can be delegated, via an intra-chip Router. By comparison with a dual Pentium system where there are 2 of every processing resource including expensive and underutilized mathematical processing logic, it makes clear that the more efficient balance is to have numerous IPUs executing the bulk of the instruction set themselves, e.g., executing simple integer processing instructions, and delegating other instruction processing to the proper processing resources, which can be shared and should be supplied in quantities appropriate to their utilization to avoid underutilization.*

**[0056] IPUs 102, 104, 106, 108 are the general integer and logic processing units of Processor 100 and each of the IPUs is responsible for running a separate program thread (i.e., a part of a program that can be executed independently of other IPUs). Furthermore, each of the IPUs may run independent processes and program software.** In one embodiment, each IPU may be a simple 32 bit microcomputer with 8.times.32 bit data registers (D0-7), 8.times.32 bit address registers (A0-7), a 32-bit instruction pointer (IP) and a 32 bit return link pointer (RP). Address register A7 may be used as a stack pointer. **Alternatively, each IPU may be a 32-bit or 64-bit microcomputer.** A 32-bit IPU may have a bank of 16 general purpose registers of 32 bits, organized as 8 address registers and 8 data registers. In a 32-bit implementation of Processor 100, 64 bit operations may take an extra cycle (i.e., double precision operations). A 64-bit IPU may have a bank of 16 general purpose registers of 64-bits, organized as 8 address registers and 8 data registers. *This paragraph describes that IPUs are each responsible for independently running a separate program thread and that they are in fact computer processor cores with registers including an instruction pointer.*

**[0057] Each of IPUs 102, 104, 106, 108 is configured to execute the instructions of a thread and/or process and perform relatively simple calculations of the instructions such as add, subtract, basic logic (e.g., Boolean operations) and/or integer calculations. Each of the IPUs are further configured, upon decoding an instruction while executing the instructions of the thread and/or process, to send calculation requests along with data and opcode of the instruction to MPU 114 for relatively complex calculations such as multiplication, division and/or floating point calculations. Alternatively, the IPUs may be configured to send the instructions themselves that require complex calculations to the MPU without decoding the instructions.** Similarly, each of the IPUs may also be configured, while executing the instructions of the thread and/or process, to send access requests to L2 cache 112 for accessing data and/or instructions stored in the L2 cache. ** *This paragraph describes the IPUs as each decoding and executing the instructions of a program thread, (described elsewhere as accessed via requests to L2 cache), directly performing simple integer calculations and, on encountering an instruction that requires it, requesting the MPU to perform the relatively complex calculation for that specific instruction. Note, as is also described elsewhere, memory data access is also by an IPU request to L2 cache.*

*Paragraph [0058] describes that the example IPUs may each have an L1 Instruction and data cache. Nowhere in the specification are MPUs described as having L1 cache at all.*

*Paragraph [0059] describes that the Router is typically a cross point switch with an arbitrated request/acknowledge mechanism. This is very low latency circuitry to switch system buses at instruction execution speed.*

*Paragraph [0060] describes that the L2 cache may be conventional.*

*Paragraph [0061] describes that the L2 cache may also be enhanced with a built in arithmetic logic unit that can perform atomic read-modify-write instructions such as lock/unlock.*

[0062] **MPU 114, one of the other shared resources in Processor 100, is a complement processing resource for IPUs 102, 104, 106, 108 and is configured to perform relatively complex calculations of instructions such as multiply, divide and/or floating point calculations upon receiving requests from the IPUs. The MPU is configured to receive calculation requests from the IPUs with decoded opcodes and data via Router 110 along with data bus 140. Alternatively, the MPU may be configured to receive and execute encoded instructions with complex calculations forwarded from the IPUs.** The calculation results are delivered via return data bus 148 to the requesting IPUs. The MPU may be lightly pipelined to deal with multiple requests from the IPUs before the completion of a request from an IPU (i.e., accepts new requests from the IPUs while others are in progress). For example, the MPU is pipelined with separate short pipes for floating point add/subtract, and integer/floating point multiply/divide operations. A state machine implements multi-sequence iterative functions such as integer/floating point divide, square root, polynomials and sum of products. These in turn may be used to efficiently implement more complex functions. *Thus, paragraph [0062] describes that the MPU is a shared complementary processing resource for IPUs and performs relatively complex calculations upon receiving requests from the IPUs. The example MPU has separate short pipelines for floating point add/subtract, integer and floating point multiply and divide operations. They only execute specific individual instructions and only when requested to do so by an IPU with the instruction and operand data supplied by the IPU, and a result is returned to the IPU. No mention is made of MPUs receiving instructions in any way other than through delegation from an IPU. As there can be more than 1 MPU and they can be different, each of the separate pipelines in the example MPU can be separate MPUs if desired.*

**[0063] In one non-limiting implementation of Processor 100, a single MPU (e.g., MPU 114) is shared by multiple IPUs (e.g., IPUs 102, 104, 106, 108) using the silicon area more effectively when compared to a conventional processor with a similar size.** The streamlined architecture of the IPU structure (e.g., 32 bit microcomputer) makes it practical to incorporate several of the IPUs on a single chip. Current conventional designs become even more inefficient in contrast to Processor 100 of the present invention as the effective area of silicon required becomes even greater when such conventional processors are duplicated and employed as monolithic structures in parallel. *This paragraph describes that the example 4 IPUs sharing 1 MPU Processor, needs only ¼ of the MPU circuitry of a conventional 4 core multiprocessor, uses the silicon area more effectively and thus makes it practical to have more IPUs than conventional processors in the same area of silicon.*

*Paragraph [0064] describes that, because more IPUs than conventional processors can fit in the same area of silicon, there is a possible tradeoff where the IPUs together can have the same or higher performance while not being clocked at so high a clock frequency. As performance scales linearly with clock frequency and number of IPUs while power energy consumption scales by the much higher cube law with clock frequency, this can save a lot of energy and greatly lower heat problems.*

*Fig. 3 of the application shows the input and output connections of an example computational type resource, noted above as called an MPU, showing it as a pipelined computational processor that can accept a delegated instruction from the router with parameters from one of multiple First Type resources in every clock cycle, returning the result after pipelined computation. Examples of different delegated computations are also shown.*

*Paragraph [0065] describes Fig. 3 which is a simplified circuit diagram of an example of the pipelined data paths at the input and output connections of an example MPU.*

**[0066]** If the arithmetic operation required is more complex than integer add, subtract or compare, or floating point sign/magnitude compare, the operation may not be executed by the IPU. Instead the B and C parameters of the complex arithmetic operation are loaded into the operand pipeline registers and **a request issued for access to the shared MPU**, which can accept a new operation into its pipeline or pipelines every clock cycle. If more than one IPUs request MPU service in the same cycle, the request from the IPU which is running at the highest priority is granted. The requests from the IPUs are pipelined with pipeline input registers. A decoder decodes the requests determining the requesting IPU number. The decoded number of a requesting IPU may be added to the output result at the pipeline output register. In the execute pipeline phase, **an IPU may issue a request to the Router to connect to the shared MPU to execute, for example, a complex math instruction.** On the next free cycle, the IPU may write the function parameters to the input registers of the MPU. If the instruction requires double precision, the writing of the function parameters may take two cycles. **When the MPU has calculated the result, it returns the result on the dedicated return bus to all IPUs, with the IPU number of the corresponding request so the correct IPU will accept it.** If the result is double precision, this may take two cycles as well. *Thus, this paragraph describes the pipelined cycle by cycle priority arbitration request/acknowledge mechanism of the example processor. An IPU requests the Router to grant access to an MPU and, if granted, writes the instruction and parameters via the Router to the MPU's pipelined input registers along with the IPU's ID number so the computed result will be returned to the correct IPU because of the sharing of the MPU.*

*Paragraph [0067] describes that the circuitry of an MPU is simplified because there is no relationship between successive operations in the MPU.*

*Paragraph [0068] describes the Processor instruction set as having spare instructions for more kinds of computational processing resources than the example MPU, including spare expansion capacity for future needs.*

*Paragraph [0069] describes Fig. 4 and Fig. 6 as a simplified circuit diagram of an example of the pipelined data paths and instruction decode path of an example IPU. MPUs are never described as having the ability to fetch instructions from memory.*

*Fig. 4 shows the internal register and computation pipeline of an example first type processing resource, showing that it fetches parameters from a register or registers and either handles them internally or has them routed to the inputs of one or more computational resources. The result from either the internal processing or routed from the outputs of the computational processing resources are written back to a register or registers.*

*Fig. 6 shows the instruction fetch from memory via L1 cache and the instruction decode pipeline of a first type resource. A computational processing resource lacks this.*

*Paragraphs [0070-0087] describe internal details of the example IPU.*

**[0077]** Each of Branched Data Buses are connected to Router 110 for transmitting a request to the shared resources (e.g., L2-cache 112 and MPU 114). **For example, the operand and/or data of an instruction that requires a complex calculation may be sent to the shared resources** (e.g., MPU 114) before the ALU of the IPU performs the calculation.

*Paragraph [0088] describes Fig. 8 as a simplified circuit diagram of the pipelined data paths of an enhanced L2 cache that implements atomic memory operations.*

*Fig. 8 shows an innovative example computational processing resource that connects to the shared memory and other devices via an L2 cache, showing it as a pipelined computational processor that can accept from the Router a delegated instruction with parameters from one of multiple first type resources in every clock cycle, returning the result after pipelined computation.*

*Paragraph [0089] describes in detail how the addition of a simple arithmetic logic unit pipeline stage to the data path of an. L2 cache, with appropriate control circuitry, creates an enhanced L2 cache (112) usable to implement atomic memory read-modify-write instructions such as lock, unlock or semaphore much more efficiently than conventional spinlocks, locking the L2 cache only briefly compared to the bus locking to the IPU and L2 cache thrashing of a conventional spinlock.*

*Paragraph [0090] describes how the atomic lock and semaphore instructions can be used with the described hardware to control access to shared data without the need for L1-L2 cache coherency with its very high overheads and delay.*

**[0100] In every clock cycle, every IPU may send a request to one of the shared resources (i.e., L2 cache 112 and MPU 114).**

**[0104]** FIG. 9 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure **providing a resource delegation process in IPUs 102, 104, 106, 108 of Processor 100 in which a requesting IPU constructs and delegates a request to other processing resources (e.g., L2 cache 112 and/or MPU 114).**

**[0105] The general operation iterative flow for a delegation process in one of an IPU (e.g., IPU 102) may be summarized as: (1) the constructing of a request if necessary to a delegated resource, (2) routing the request, and (3) obtaining a response from the delegated resource at the requesting unit.** At some point, an instruction is fetched for the requesting IPU 903. Upon obtaining an execution-instruction, a determination is made by the IPU if the instruction is intended for execution on one of the other processing resources (e.g., L2 cache 112 and/or MPU 114) based on the decoding of the opcode of the instruction 905. For example, if the opcode indicates that an adding or subtracting calculation is required, and no information is required from the external cache (i.e., L2 cache 112), the instruction then may be executed in the IPU without delegation 907. **If, however, the opcode indicates that at least one of the other processing resources is required to service the instruction** (e.g., a multiplication or division calculations, or information from the L2 cache is required for the execution), then a request may be constructed by the IPU 909. For example, in constructing the request, the IPU may combine several information such as the IPU number, decoded opcode, values from the status registers and priority bit status registers (e.g., lock/unlock/interrupt hardware priority bit). Subsequently, the IPU provides the constructed request to Router 110 911. Upon providing the request to the Router, the IPU is put to sleep and waits for a reply or for a request for the IPU to wake 913.

**[0177]** FIG. 25 is of a flow diagram illustrating one, non-limiting example, inventive aspect of the present disclosure providing a wait-on-semaphore instruction facility. A wait-on-semaphore instruction is an atomic operation for a processing cache (e.g., L2 cache 112). At some point, a wait-on-semaphore instruction is fetched 2501 and **used to claim exclusive use of some shared resource** or memory 2503. Upon interpreting a wait-on-semaphore instruction operation 2505, the data optimizing cache will subtract "1" from a semaphore variable in the L2 cache 2507. If the processing cache determines that the semaphore variable is negative 2509, an operating system trap-call to wait on the queue is issued 2513. **This trap-call causes rescheduling so that other program threads/processes may wait for execution of a thread to release its semaphore**

**on a particular IPU.** If the semaphore variable is not negative 2509, the processing cache may continue to operate 2511.

Based upon the above, it is respectfully submitted that the claims are all entirely supported. As can be seen from the description, the only way that the computational processing resources receive instructions is through delegation by an IPU. Nothing in the specification describes or hints in any way that those computational resources receive instructions any other way.

In view of the above, withdrawal of the Section 112 rejections asserting lack of written description is respectfully requested.

***Response to Section 103 Rejections***

Claims 144, 155-156, 164-165 and 167-170 have again been rejected for obviousness over a combination of the previously cited Bonola and Bridges references. Claim 144-145 and 171-172 have been rejected again for obviousness over a combination of the previously cited Butterworth and Bridges references. Claim 157-158, 160 and 1615 have again been rejected for obviousness over a combination of the previously cited Bonola, Bridges and Mohamed art.

Applicant respectfully traverses the obviousness rejections for the following reasons.

The Final Office Action in paragraph 33 has challenged Applicant's assertion that none of the Bonola slave processors are what Applicant has called "second type" or now "computational type" processing resources and asserts, without citation, that those slave processors only execute delegated instructions. The Final Office Action flatly ignores the passages from Bonola cited in response to the previous Office Action that establishes Applicant's position and refutes paragraph 33 of the Final Office Action.

In Bonola, each microprocessor fetches and executes an instruction stream from memory, and inter-processor communication among the microprocessors is by software convention. This is evident from Bonola at: Fig 1; Col. 3, Lines 16-26 and 45-51; Col. 4, Lines 43-47; Col. 7, Lines 7-15; and Col. 8, Lines 55-61. Moreover, in Bonola, contrary to the assertion in the Final Office Action, program instructions are <u>not</u> delegated by any processor of Bonola to any slave or other processor in Bonola for execution, commands are passed, but they are not from the program instruction thread the "master" is executing. In other words, there is no offloading of program instruction execution. This is evident from Bonola at: Col. 3, Lines 37-42 and 45-51;

and Col. 8, Lines 55-61. The passing of commands from master to slave in Bonola is not the passing of program instructions from a thread that the master is executing to a shared slave for execution.

The Final Office Action assertion is incorrect and makes no attempt to address the above-cited passages of Bonola that establish the contrary.

If the Patent Office disagrees again, Applicant respectfully requests that it point out exactly why those passages do not refute the Patent Office position or expressly quote a passage from Bonola that clearly establishes that the slaves receive delegated program instructions as noted above and not merely control instructions.

**US Patent 5,706,514 (Bonola).**

As established by the passages cited above Bonola does not have the recited shared computational processing resources of the pending claims. Thus, Bonola cannot disclose or hint at any form of method involving delegating program instructions from a thread to such a shared elemental processing resource for execution.

**US Patent 6,081,860 (Bridges et al.)**

Notwithstanding the Final Office Action re-iteration of the rejections based upon Bridges et al., the Final Office Action tacitly accepts Applicant's position that Bridges describes a multiprocessor system made up of multiple microprocessors that connect to multiple memory controllers or devices via a shared, arbitrated Processor Local Bus (PLB). The bus arbiter can accept a memory request while another is in progress, pipelining the address of the next request so data transfer can begin immediately after the conclusion of the previous request. It refers to the processors as "master devices" and the memory controllers or devices as "slave devices", but it is a memory address that is pipelined and memory data that is transferred, so it is clear that the "slave" is not a slave processor as the Office Action asserts. See, e.g. Bridges at: Figs 1-5; Col. 2, Lines 47-52; Col. 4, Lines 9-12 and 49-62. Each "slave device" in Bridges et al. is a memory controller or a device that does not execute the instructions transferred, it simply reads and writes memory in the usual ways but additionally incorporates address pipelining. See, e.g. Bridges at: Fig 1; Col. 2, Lines 47-52; Col. 4, Lines 9-12 and 49-62. There simply is no delegation of instruction from a thread the "master devices" are executing to the devices that are "slave" devices. Thus, Bridges lacks shred computational processing resources and, hence cannot disclose or hint at any form of method involving delegating program instructions from a thread

being executed by a first type processing resource to such a shared elemental processing resource for execution.

**US Patent 6,907,454 (Butterworth et al.)**

Butterworth et al. describes a dual processor system with a high performance master processor connected, via a bridge circuit, to a lower performance slave processor and various high and low speed memory busses.

The specific processors mentioned in Butterworth et al. are an IBM PowerPC 705 master processor and a PowerPC 403 slave processor that are connected to each other, and to memory, by a bus subsystem. It is therefore clear that the slave processor is not fundamentally different from the master processor, but is merely of lower performance. [See, e.g. Butterworth at: Fig 1; Col. 3, Lines 14-18 and 23-27; and Col. 4, Lines 29-33]

The high performance master processor uses software commands passed via a bus subsystem to the slave processor in order to delegate to software drivers on the slave processor access to memory and devices on the high and slow speed memory busses. See, e.g. Butterworth at: Figs 4-5; Col. 3, Lines 14-18 and 23-27; Col. 4, Lines 43-48; Col. 5, Lines 7-9; and Col. 6, Lines 45-63.

Butterworth et al. is simply a dual processor system, it lacks the requisite multiple master processors which, by definition, are necessary for sharing of another processor between them.

Each processor in Butterworth et al. would best be equated as an example of a claimed elemental processing resource of a "first type," irrespective of whether it is a master or slave. This is clear because each processor in Butterworth et al. fetches and executes an instruction stream from memory and inter-processor communication is by software convention, with commands and data passed via a bus subsystem. See, e.g. Butterworth at: Figs 1, 3-5; Col. 3, Lines 14-18; Col. 4, Lines 43-48; Col. 5, Lines 7-9; and Col. 6, Lines 45-63.

Butterworth et al. does not, indeed cannot, disclose or hint at any form of method involving any of two processors sharing a third computational resource to which either can delegate instructions for execution.

**US Patent 5,978,838 (Mohamed et al.)**

Mohamed et al. describes a dual processor system on a chip where the two processors each execute different instruction sets. See, e.g. Mohamed at: Fig 1; Col. 2, Lines 8-9 and 31-34; and Col. 3, Lines 32-35. One of the two processors is a general purpose processor and the other

is an SIMD vector processor. See, e.g. Mohamed at: Fig 1; Col. 2, Lines 31-34; Col. 3, Lines 32-35. Each processor in Mohamed has its own discrete instruction and data caches (i.e. both processors are fully independent peer processors that each execute their own software from memory). Specifically, each processor independently fetches and executes a completely different instruction stream from memory and inter-processor communication is by software convention, with commands and data passed via registers and interrupt signals in bridge circuitry. See, e.g. Mohamed at: Fig 1; Col. 1, Lines 61-67; Col. 2, Lines 66-67; Col. 3, Lines 1-2 and 37-43. All that the two processors have in common is a main memory bus and interface registers. See, e.g. Mohamed at: Fig 1 and Col. 3 Lines 15-20.

Mohamed et al. is simply a dual processor system, so necessarily there can be no sharing by two processors of any other processor(s).

Mohamed et al. does not, indeed cannot, disclose or hint at any form of method involving delegating instructions to a second type elemental processing resource, let alone a method that requires at least two first type elemental processing resources sharing at least two second type elemental processing resources.

**Claim 144 Is Allowable**

Amended claim 144 recites, *inter alia*, "the multiple elemental processing resources comprising two or more elemental processing resources of a first type each sharing two or more elemental processing resources of a pipelined computational type with at least one other of the two or more processing resources of the first type" and "the processing resources of the first type each being configured to receive a non-delegated program instruction from a thread in memory that it can execute or delegate for execution by one computational type elemental processing resource" and further recites "the processing resources of the computational type each being configured to only receive and execute instructions that were delegated to their pipeline for execution by processing resources of the first type and by being shared by at least two elemental processing resources of the first type . . ."

Since this structure is neither disclosed nor fairly suggested by any combination of the cited references, no method performed in such a configuration would be rendered obvious by any of the cited references.

Moreover, amended claim 144 expressly recites as part of the method "obtaining an execution instruction from the thread, wherein the execution instruction is obtained at one of the

elemental processing resources of the first type; determining whether an operation-code within the execution instruction is incapable of being executed by the one elemental processing resource of the first type and thus should be delegated to one elemental processing resource of the computational type that the one elemental processing resource of the first type shares in common with another elemental processing resource of the first type;" and further recites "if the execution instruction should be delegated, providing a specific elemental processing resource of the first type access to a specific computational type elemental processing resource it shares with another of the elemental processing resources of the first type based upon an arbitration request by the specific elemental processing resource of the first type; and routing the execution instruction and an identifier for the specific elemental processing resource of the first type that delegated the execution instruction to a pipeline of the specific elemental processing resource of the computational type that is shared in common and is capable of executing the operation code."

As evidenced by the passages cited for each of Bonola, Bridges, Butterworth and Mohamed, none of those cited references, alone or in combination, disclose, suggest, or hint at a system of processors configured as in claim 144, nor a configuration that operate as claimed in that claim.

Accordingly, it is respectfully submitted that claim 144 is non-obvious and allowable over the various combinations of the cited references.

**The Claims Depending From Claim 144 Are Allowable**

Claims 145, 155-158, 160, 164-165, and 167-172 all depend, directly or indirectly, from claim 144, so they are all allowable for at least the same reasons.

Moreover, the dependent claims add additional aspects that are lacking from the cited references as well, taken alone or in combination. Accordingly, those dependent claims are independently allowable.

For example, dependent claims 158, 160 and 164 respectively each specify at least one specific "flavor" for one or more of the computational processing resources. The cited references do not disclose or suggest any such shared computational processing resources to which program instructions can be delegated for execution as defined in the claims at all, they necessarily cannot disclose such particular resources.

In addition, claims 158, 160 and 164 allow for both homogeneity or heterogeneity among the computational processing resources. This aspect is similarly lacking from all of the references.

Dependent claim 167 requires, while acting on instructions from an individual thread, that a first type resource to sleep while a second type resource executes an instruction from the thread delegated to it. No such operation is remotely hinted at by the cited references.

Dependent claim 168 specifically requires the resources "dynamically to turn on and off to maintain a desired level of power draw while maximizing processing throughput." The Office Action-cited part of Bonola does not begin to disclose or suggest this aspect. All Bonola discloses is that a processor can "sleep" which is a far cry from regulating sleep in the manner claimed.

Similarly, Bonola's simple disclosure of a processor that can sleep does not begin to disclose or suggest the subject matter of claims 169 and 170 which respectively require "generating an execution-instruction signal from at least one of the elemental processing resources, wherein the execution-instruction signal from the elemental processing resources themselves shuts off processing resources while idling" and "generating an execution-instruction signal from at least one of the elemental processing resources, wherein an execution-instruction signal from processing resources themselves turn on processing resources when execution instruction signal processing is required."

Accordingly, the above-referenced dependent claims are all additionally allowable for those independent reasons.

**Claim 1615 Is Allowable**

Amended claim 1615 now recites: a method of execution-instruction delegation among multiple interdependent elemental processing resources, the processing resources comprising at least two Instruction Processing Units (IPUs) and two or more computational processing resources comprising, in any combination, one or more of MPUs, instruction delegating caches, encryption or decryption logic or vector processors, the multiple elemental processing resources being configured to perform processing operations according instructions in an instruction set but being individually incapable of servicing at least one instruction in the instruction set, the elemental processing resources being configured such that a first and a second of the IPUs each share at least two of the computational processing resources in common, the method comprising:

receiving, from memory, one of multiple execution-instructions of a thread at a first Instruction Processing Unit (IPU) of the at least two IPUs;

processing the first execution-instruction using the first IPU;

receiving an other execution-instruction of the thread at the first IPU;

determining that the other execution-instruction from the thread can not be processed by the first IPU and must be executed by a computational processing resource;

receiving a request by the first IPU for access to one of the shared-in-common computational processing resources that can execute the other execution instruction;

receiving an acknowledgement that allows delegation by the first IPU to the one of the shared-in-common computational processing resources;

delegating the other execution-instruction from the thread to a pipeline for the one of the two or more shared-in-common computational processing resources along with an identifier of the first IPU so that it can process the other execution instruction from the thread and return a first result of the execution to the first IPU based upon the identifier;

maintaining the thread in the first IPU in a sleep state until an indicator that the processing of the other execution-instruction from the thread by the computational processing resource is returned to the first IPU based upon the identifier;

receiving an execution instruction of a different thread at the second IPU;

determining that the execution-instruction of the different thread can not be processed by the second IPU but can be processed by the one shared-in common computational processing resource that is shared between the second IPU and the first IPU;

arbitrating access to the one shared-in-common computational processing resource by the second IPU; and

delegating the execution-instruction from the different thread to the one shared-in-common computational processing resource along with an identifier of the second IPU so that the one shared-in-common computational processing resource can execute the execution-instruction from the different thread and return a second result to the second IPU based upon the identifier of the second IPU.

The structure recited in amended claim 1615 is neither disclosed nor fairly suggested by any combination of the cited references, consequently no method performed in such a structure would be rendered obvious by any of the cited references.

Moreover, none of the cited references, alone or in combination, disclose or suggest the specific method steps recited in claim 1615. Accordingly, amended claim 1615 would not have been obvious in view of any combination of the cited references either.

## CONCLUSION

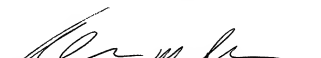
It is respectfully submitted that all of the pending claims would not have been obvious in view of the references cited by the Final Office Action, whether taken alone or in combination, and the objections have been overcome. Accordingly, it is respectfully submitted that all of the pending claims are allowable and early, favorable action in that regard is respectfully requested.

## AUTHORIZATION

The Commissioner is hereby authorized to charge any additional fees which may be required for consideration of this Amendment to Deposit Account No. **504827**, Order No. **1004437-006US**.

Respectfully submitted,
Locke Lord Bissell & Liddell, L.L.P.

Dated: February 1, 2011

By: 

Alan M. Sack
Reg. No. 31,874

Address Associated With Customer Number:
**85775**
(212) 415-8600 Telephone
(212) 303-2754 Facsimile